MuCool ModBus Protocol

Local application

Tue, Dec 4, 2007

The liquid helium refrigerator to be used for MuCool in MTA uses hardware that can be accessed via an RS232/RS485 protocol. This note describes the support for handling this protocol from a front end so that the data collected can become part of the Acnet-accessible data pool. The method uses a local application called `MODB` to issue requests for data and interpret the ensuing replies.

The RediStart Micro II, from BenShaw, supports a serial protocol at bit rates of up to 9600 Baud. We plan to use that rate in this implementation, so that we have byte rates of about 1 KHz. The interface operates with 8 data bits, one stop, no parity, and no handshaking. Whether RS485 is used will not affect the software support, as the start of each message is always the target address. If too much time (3.5 byte times) passes without receiving any data bytes, the receiver can assume that no more bytes are to come in the present message. This can help avoid "getting out of sync."

Up to now, the serial port support in the 68K-based IRM front ends assumes Ascii codes are used. But to support this new protocol, full binary support is needed. There can be no special checking for a `CR` code (`0x0D`), for example, to detect the end of a line of text. All 8-bit patterns must be accepted as binary data. To support this, we need to add an ability to handle serial I/O in Binary mode. Another note, *Binary Serial Data*, describes how to do this.

ModBus protocol

The data to be accessed from the RediStart Micro II is treated as two sets of registers, and each register has a separate address. A function code selects the register set, and a 12 bit register offset is passed in the protocol. Here is the set of bytes that comprise a request for register values:

Field	*Size*	*Meaning*
address	1	target node address (the default factory setting is `0x7F`)
function	1	function code, either `0x03` (read holding regs) or `0x04` (read input regs)
offset	2	register offset, `0—0x0FFF`
nregs	2	number of sequential registers to access
crc	2	CRC-16 cyclic redundancy check code

A reply message has the following format:

Field	*Size*	*Meaning*
address	1	target node address
function	1	function code, either `0x03` or `0x04`
nbytes	1	#data bytes to follow = twice the number of registers requested
data	2n	register data values, where n = nregs in request
crc	2	CRC-16 cyclic redundancy check code

The maximum number of registers that can be read with one request is 60, so that the maximum length of a reply message is 125 bytes.

Only a single request can be active. Sending a request at rates up to 15 Hz, we could obtain up to 30 words of data in response, given the 9600 baud serial rate. Of course, there is no need for collecting this data at 15 Hz; one could send requests more leisurely. Also, if the data needed is not all reasonably accessible with a single request for sequential registers, it might take at least one 15 Hz cycle per request. For this application, it is not thought that quick access to this data is required. It is also not thought necessary that it be correlated with usual 15 Hz data pool readings, either. All this thinking is quite normal when accessing data via a serial port.

Parameter layout

Consider the following parameter list for MODB:

Field	*Size*	*Meaning*
ENABLE B	2	Usual local application enable Bit#
WAIT CYC	2	#cycles delay between requests
TST ADDR	2	Test unit address
FUNCTION	2	Function code 3 or 4
OFFSET	2	Register offset
NREGS	2	Number registers to access

This can help during initial testing, but in order to be flexible in specifying what registers should be collected, assume the real specs are included in a data file DATAMODB. Each spec is 4 words:

address	Target unit address
register	Register number (such as holding register 40001 decimal)
nregs	#registers
channel	target Chan# for reply data words

We can enable the use of the test parameters by the value of the TST ADDR parameter. When it is nonzero, the test parameters are used; when it is zero, the data file specs are used, one each cycle. To turn off the test mode, zero the TST ADDR parameter. The WAIT CYC is used for both cases.

RediStart registers

The data values that can be accessed are numerous and are described in the *RediStart Micro II RS232/RS-485 Communications Manual*. Each value is expressed as an integer in engineering units, where the units are chosen to accommodate the desired precision. As an example, the Motor FLA register has values in the range 1–1200 Amps, so that the returned values range from 1—0x04B0. Another example is the Voltage delay register, whose values range from 1–900, in units of 100 ms. This corresponds to 0.1 to 90.0 seconds. In order to map these values into our normal scale factors, the first case would use a fullscale value of 32768.0, whereas the second case might use 3276.8, if we choose to display the Voltage delay values in units of seconds. This all seems very easy.

The meaning of register offset is based upon the first register address accessible via a given function code. The address of the Motor FLA register is stated in the manual as 40001, which is the first such register listed. Access using function code 0x03 and register offset 0x0000. The Average current input register is listed as 30402. Use function code 0x04 and register offset 0x0191.

Turning to the CRC16 codes that must be used at the end of a message, a C language function by that name was obtained from the web, with text stating that it is used for "performing Modbus CRC16 generation." It employs a table of 256 data words to build the result one byte at a time, rather than one bit at a time. The LA can access this table from the first 256 words of the data file. The rest of the data file can include up to 32 four-word specs.

Operational activity

The MODB LA issues a request message, waits for the number of 15 Hz cycles given by the "WAIT CYC" parameter, and expects to find the complete reply as it calls for serial port data. If the WAIT CYC parameter is zero, the reply data should be available on the very next cycle. After the reply message is verified, including the CRC check, the data words are copied into the data pool. If the next spec address field is zero, or the end of the spec array is reached, return to the first spec.

To enable test mode, during which no other requests are sent, set the parameters accordingly,

including the nonzero address parameter. To return to operational mode, zero the address. After downloading a new version of the data file, restart the LA to make it effective.

Generic access feature

This feature permits a means of testing, at the Acnet level, an arbitrary register from the ModBus equipment, without ceasing its normal operational activity. Use a generic device, with full scale 32768.0, so that its reading is merely the binary number shown in decimal. Set a dummy channel to the register number of interest. Shortly, the value of that register should be seen in the generic device. This takes advantage of the convention that the RediStart hardware makes in scaling data to engineering units. A dummy channel holds the test register number, besides the above dummy channel that holds the reading. This is of course a single user approach, but it still can be useful. On a parameter page that displays these two values, one can verify that the register number in use is the one intended, not one that someone else set. Additional parameters can support this, each standing for a pair of sequenctial channels in which the first holds the register number, and the second holds the resulting data value.

One wrinkle here is that we need a means of specifying the address of the unit to be queried. Since we only have two units, establish two parameter sets for this. The unit number could be taken from another parameter, or it can by default 1 and 2. These are the additional parameters:

Field	*Size*	*Meaning*
`GENREG1 C`	2	Generic register# for unit 1 test Chan#
`GENREG2 C`	2	Generic register# for unit 2 test Chan#

Assume the data Chan# = register Chan# + 1. Allow for any unit addresses, assuming they are between 1 to 15, by using the high nibble in the Chan# parameter word. (The Chan# is limited to the range `0–0x07FF`.) If the high nibble is zero, assume units 1 and 2, respectively.

How can `MODB` handle it? Unlike the other test case that is supported by the earlier parameters, this one works even while `MODB` is accessing its normal data from the two units. Each time through the spec list, it acts as if there is one hidden spec, or two. Append it to the active spec list. When the end of the spec list is detected, it checks the two sets of dummy channels to fill in another spec or two. Then it advances the index to that place at the end of the spec list. Effectively, the optional generic register accesses are performed each time through the list of data file specs.

To hold a register number, we need all 16 bits. Use the 32768 full scale and 32768 offset trick to allow an effective value in a range 0–65535 for the channels that hold the number. The raw reading is assumed to have its high bit toggled when `MODB` accesses it to obtain a register number that may exceed 40000 decimal.

The result of this approach is that the user can query an individual register without going out to MTA to use the small display panel on the hardware itself.